Rapport NantralePoly

Dans ce rapport nous allons expliquer le fonctionnement des différentes classes.

Concernant les liens d’héritages, nous avons Achetable et NonAchetable qui héritent de la classe Case. De plus, Gare et Constructible héritent de la classe Achetable. Pour prendre en compte un hôtel, nous avons uniquement mis le nombre de maisons, et quand nous avons 5 maisons c’est comme si nous avons un hôtel.

Nous allons expliquer le fonctionnement des méthodes des différentes classes :

# Description des méthodes de classes

Classe Achetable :

acheter: elle prend en argument un joueur et si le joueur a assez d’argent, le prix d’achat de la case sera déduit de la fortune du joueur et l’attribut propriétaire de l’instance d’Achetable devient le joueur.

loyer : appelée lorsqu’un joueur tombe sur une case et doit payer un loyer. Cette méthode va permettre de connaître le prix du loyer à payer en fonction, soit du nombre de gares que le joueur possède (si l’autre joueur est tombé sur une gare), soit du nombre d’hôtels et de maisons si c’est une case Constructible.

reset : quand un joueur est éliminé il faut redéfinir le propriétaire de la case comme null. C’est ce que fait cette méthode.

Classe Constructible :

reset : c’est une surcharge de la méthode définie dans la classe Achetable, où l’on met à 0 le nombre d’hôtels et de maisons.

construire : Ne prend aucun argument, quand un joueur construit une maison, on retire le prix d’achat de la maison à la fortune du joueur et on ajoute 1 à nbMaison. Si nbMaison est à 5, on ne fait rien (car 5 maisons correspond à un hôtel et on ne peut pas avoir plus d’un hôtel sur une case). La vérification de si le joueur peut ou non acheter la maison à été regardé avant d’appeler la méthode.

Classe Joueur :

paiement : Cette méthode prend en argument un autre joueur. Si on fait j1.paiement(j2) alors le joueur j2 paye l’argent qu’il doit à j1 quand il tombe sur une de ses cases. Cette méthode va modifier l’attribut fortune des deux joueurs.

Classe PlateauDeJeu :

initPlateau : initialise le plateau de jeu en donnant un nom à toute les cartes ainsi que la valeur de leurs attributs.

nbGares : prend un joueur en argument, parcourt la liste des cases et compte le nombre de gares que possède le joueur.

avance : il prend en argument un entier d et une case c. On va partir de la case c, avancer de d cases dans plateau et renvoyer la case sur laquelle nous sommes tombés.

lancerPartie : cette méthode sera appelée dans la méthode main après avoir initialisé la partie et va appeler la méthode tourDeJeu jusqu’à ce que la partie soit finie. A la fin on affiche le gagnant.

joueurElimination : quand un joueur tombe sur une case d’un autre joueur et qu’il ne peut pas payer, il est éliminé. On le passe en argument, il sera supprimé de la liste des joueurs et toutes ses cases seront reset.

lancerDe : lance un dé entre 1 et 6 que l’on lancera 2 fois dans un tour de jeu. On a choisi 1-6 et pas 1-12 pour quand nous rajouterons des options (comme les sorties de prisons qui nécessitent de faire un double).

finPartie : regarde si il y a plus d’un joueur restant dans la partie. S’il en reste 1, elle renvoie True, sinon elle renvoie False.

tourDeJeu : pour chaque joueur on va effectuer les opérations suivantes :

* On va lancer 2 dés et faire la somme de ces dés
* On déplace le joueur sur la nouvelle case
* Si la case est Achetable, que le joueur a fait un lancer impair, que la case n’a pas de propriétaire et que le joueur à l’argent, il achète la case.
* Si le joueur arrive sur une case appartenant à un autre joueur, il paye ce qu’il lui doit, et s’il n’a pas assez d’argent, il est éliminé.
* Si c’est une case NonAchetable ou une case Achetable avec un lancé de dé impair et qu’il n’a pas assez d’argent, il ne se passe rien.
* Si le joueur a fait un lancé pair et qu’il possède le terrain, si il a l’argent il achète une maison.
* On regarde si il y a plus d’un joueur encore en jeu avec finPartie().
* On passe au joueur suivant.

# Organisation du travail

Au début du projet nous avons décidé de créer un coordinateur qui s’assurait de la bonne répartition de l’ensemble du travail à effectuer ainsi qu’une personne qui rassemblera tous les morceaux de code.

Nous avons utilisé un serveur discord qui nous permettait de communiquer tout en envoyant les différentes parties du code (avec les commentaires). Nous avions envisagé l’utilisation de l’outil GitHub mais nous avons abandonné l’idée car trop peu de gens savaient l’utiliser dans le groupe.

On peut résumer le projet en trois phases :

* La première consistait à tous se réunir pour construire le diagramme UML. Utiliser toutes les personnes pour le diagramme UML nous permettait de ne rien d’oublier.
* Puis, l’écriture des différentes classes du diagramme a été réalisée par binôme.
* Enfin, chaque classe a été envoyée sur le serveur discord (cf. photo ci-dessous) et une personne s’est chargée de toutes les regrouper dans le projet ‘NantralePoly’.



# Commentaires

Le premier problème est que nous n’avions pas encore terminé les spéciations (le diagramme de classes n’était pas fixe) et les gens commençaient déjà à coder.

Les gens ne pouvaient pas compiler ou tester leur code au fur et à mesure car chacun travaillait de se côté sur une classe.

Une grosse charge et une grosse pression a été concentré sur la personne qui rassemblait le code car le code était mal formé (erreur de syntaxe, erreur de nommage, spécification non suivies,…).

Finalement, le travail a été réparti de manière inégale et on un rendu qui ne fonctionne pas très bien.